

# CS273: Theory of Computation, Summer 2008

## Homework 2

Due 3:00PM Monday, June 23, 2008 at SC 3303  
Revision due 3:00PM Friday, June 27, 2008 at SC 3303

1. [15 points; 5 per part] Prove the following properties of the star operator:

(a)  $(A^*)^* = A^*$

**Solution:** If  $w$  is a string in  $(A^*)^*$ , then  $w = x_1x_2 \cdots x_n$  where each  $x_j$  is a string in  $A^*$ . Because  $x_j$  is a string in  $A^*$ , we know that  $x_j = y_{j,1}y_{j,2} \cdots y_{j,m_j}$  where each  $y_{j,k}$  is a string in  $A$ . By substitution,  $w = (y_{1,1} \cdots y_{1,m_1})(y_{2,1} \cdots y_{2,m_2}) \cdots (y_{n,1} \cdots y_{n,m_n})$  and so  $w$  is the concatenation of strings in  $A$ . Therefore  $(A^*)^* \subseteq A^*$ .

Conversely, if  $B$  is any set of strings, then  $B \subseteq B^*$  because each string  $w$  in  $B$  is clearly the concatenation of zero or more strings in  $B$ . Setting  $B = A^*$ , we see that  $A^* \subseteq (A^*)^*$ . ■

(b)  $A \circ (A^*) = (A^*) \circ A$

**Solution:** Both sides are equal to  $\{x_1 \cdots x_n \mid n \geq 1 \text{ and each } x_j \text{ is in } A\}$ . ■

(c)  $\{\varepsilon\} \cup (A \circ (A^*)) = A^*$

**Solution:** As in part (b),  $A \circ (A^*)$  is the set of all strings that are the concatenation of one or more strings in  $A$ . It is also true that  $\{\varepsilon\}$  is the set of all strings that are the concatenation of exactly zero strings in  $A$ . Hence,  $\{\varepsilon\} \cup (A \circ (A^*))$  is just the set of all strings that are the concatenation of zero or more strings in  $A$ , which is the definition of  $A^*$ . ■

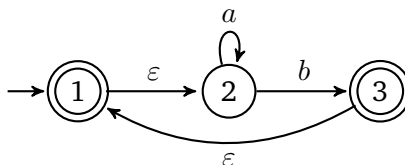
2. [10 points] Prove or disprove: All finite languages are regular.

**Solution:** This is a true statement; we will prove it. Let  $A$  be a finite language. Because  $A$  is finite,  $A = \{w_1, w_2, \dots, w_n\}$  for some strings  $w_j$ . Consider the regular expression  $R = w_1 \cup w_2 \cup \cdots \cup w_n$ . Because the language of  $R$  is  $A$ , we see that  $A$  is regular. (What would go wrong with this proof if  $A$  were not finite?) ■

3. [20 points; 10 per part] For each of the following regular expressions over  $\Sigma = \{a, b\}$ , give an equivalent DFA:

(a)  $(a^*b)^*$

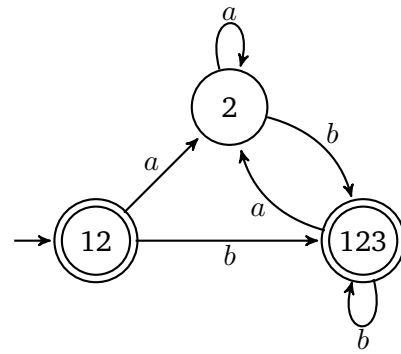
**Solution:** The following NFA recognizes the language of the given regular expression.



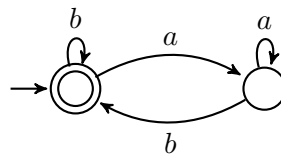
Converting the NFA to a DFA, we obtain the following transition table and its corresponding state diagram.

$\delta$	$a$	$b$
$\rightarrow \{1, 2\}$	$\{2\}$	$\{1, 2, 3\}$
$\{2\}$	$\{2\}$	$\{1, 2, 3\}$
$\{1, 2, 3\}$	$\{2\}$	$\{1, 2, 3\}$

The accept states are  $\{1, 2\}$  and  $\{1, 2, 3\}$ .



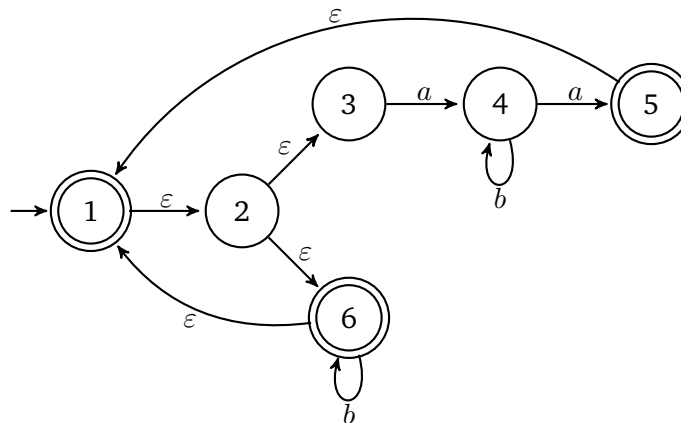
As some students have noticed, the states  $\{1, 2\}$  and  $\{1, 2, 3\}$  can be combined to give a simpler DFA.



The language of this regular expression is therefore  $\{w \mid w = \epsilon \text{ or } w \text{ ends in } b\}$ . ■

(b)  $(ab^*a + b^*)^*$

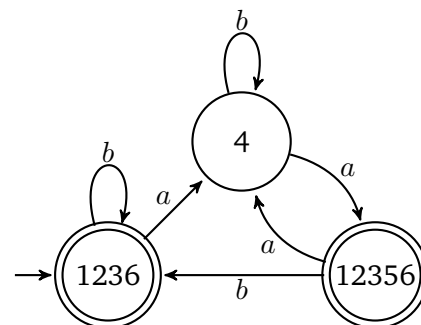
**Solution:** The following NFA recognizes the language of the given regular expression.



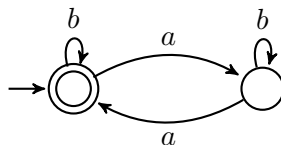
Converting the NFA to a DFA, we obtain the following transition table and its corresponding state diagram.

$\delta$	$a$	$b$
$\rightarrow \{1, 2, 3, 6\}$	$\{4\}$	$\{1, 2, 3, 6\}$
$\{4\}$	$\{1, 2, 3, 5, 6\}$	$\{4\}$
$\{1, 2, 3, 5, 6\}$	$\{4\}$	$\{1, 2, 3, 6\}$

The accept states are  $\{1, 2, 3, 6\}$  and  $\{1, 2, 3, 5, 6\}$ .



Once again, as some students have noticed, this DFA can be simplified by combining states  $\{1, 2, 3, 6\}$  and  $\{1, 2, 3, 5, 6\}$ .



Therefore the language of this regular expression is  $\{w \mid w \text{ contains an even number of } as\}$ . ■

4. [15 points] For each language  $A$  over  $\Sigma = \{0, 1\}$ , define

$$\text{ONEAWAY}(A) = \{w \mid \exists y \in A \text{ such that } y \text{ is obtained by flipping exactly one bit in } w\}$$

For example,  $\text{ONEAWAY}(\{01, 111\}) = \{00, 11, 110, 101, 011\}$ . Prove that if  $A$  is regular, then so is  $\text{ONEAWAY}(A)$ .

**Solution:** Let  $A$  be a regular language. Because  $A$  is regular, we know that there is a DFA  $M = (Q, \Sigma, \delta, q_0, F)$  that recognizes  $A$ . We show how to convert  $M$  to an NFA  $N$  that recognizes  $\text{ONEAWAY}(A)$ . Let  $w$  be a string in  $\Sigma^*$ . How can  $M'$  test if  $w$  is one bit flip way from a string in  $A$ ?

The main idea is to have  $N$  contain two copies of  $M$ . First,  $N$  simulates  $M$  using the first copy of  $M$  for some number of steps. Then, at some point,  $N$  guesses that the next character it reads is the one that is flipped; it reads this symbol and transitions to the second copy of  $M$ . It stays in the second copy of  $M$  throughout the remainder of the computation. All of the accept states of  $N$  are contained in the second copy of  $M$ , which ensures that  $N$  accepts the strings that are one bit flip away from strings in  $A$ .

Formally, we define  $N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  as follows.

- (a)  $Q_N = Q \times \{\text{PRE-FLIP}, \text{POST-FLIP}\}$ . We use  $\{(q, \text{PRE-FLIP}) \mid q \in Q\}$  for the states of the copy of  $M$  that has control before guessing the flipped bit and  $\{(q, \text{POST-FLIP}) \mid q \in Q\}$  for the states of the copy of  $M$  that has control after guessing the flipped bit.
- (b) We define  $\delta_N$  as follows.

$$\begin{aligned} \delta_N((q, \text{PRE-FLIP}), a) &= \{(\delta(q, a), \text{PRE-FLIP}), (\delta(q, \bar{a}), \text{POST-FLIP})\} \\ \delta_N((q, \text{POST-FLIP}), a) &= \{(\delta(q, a), \text{POST-FLIP})\} \end{aligned}$$

(Here,  $\bar{a}$  is the bit that is complementary to  $a$ , so that  $\bar{0} = 1$  and  $\bar{1} = 0$ .)

- (c)  $q_N = (q_0, \text{PRE-FLIP})$
- (d)  $F_N = \{(q, \text{POST-FLIP}) \mid q \in F\} = F \times \{\text{POST-FLIP}\}$ .

■

5. [30 points; 10 per part] Give a regular expression for each of the following languages over  $\Sigma = \{0, 1\}$ :

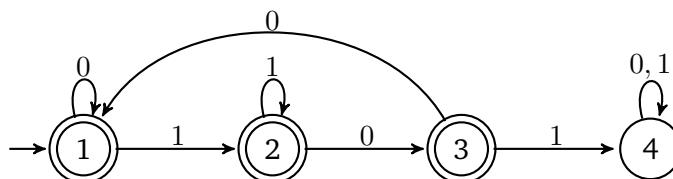
- (a)  $\{w \mid w \text{ does not contain the substring } 10\}$

**Solution:** If  $w$  does not contain the substring 10, then every 0 in  $w$  must occur before every 1 in  $w$ . Therefore  $0^*1^*$  is a regular expression for this language. ■

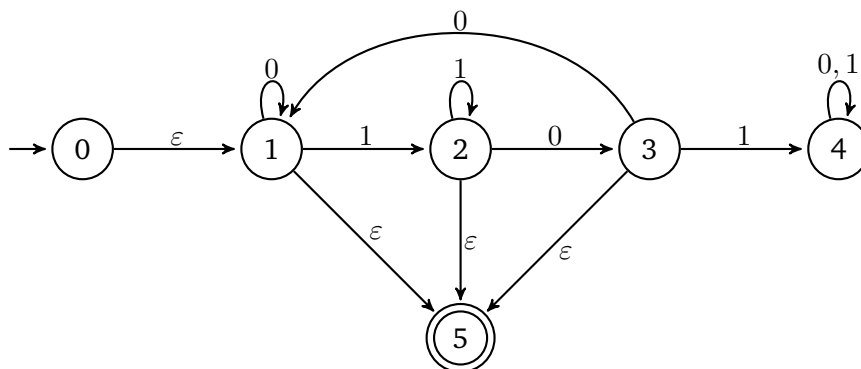
(b)  $\{w \mid w \text{ does not contain the substring } 101\}$

**Solution:** Let  $A$  be the language above. Note that  $A$  is too complex for us to design a regular expression directly. Instead, we construct an DFA and convert it to a regular expression.

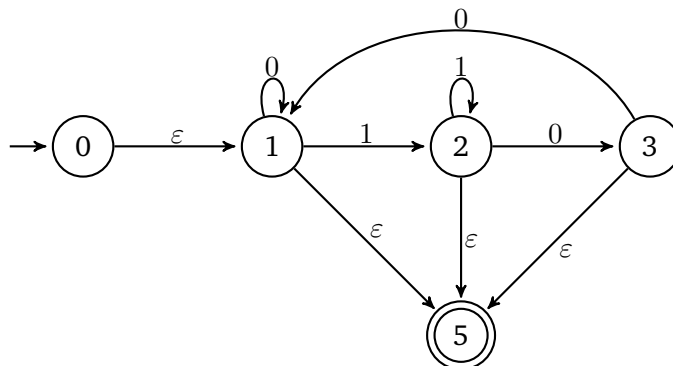
The following DFA recognizes  $A$ .



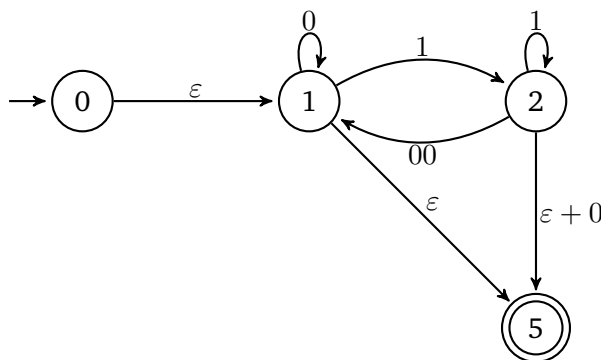
The first step is to convert to an GNFA with a new start state (labeled 0) and a single, new accept state (labeled 5).



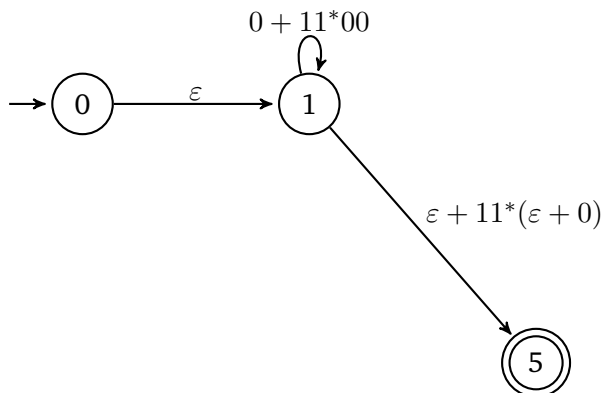
Next, we rip out states 1 through 4 one by one, each time converting to a smaller equivalent GNFA. The algorithm will give us a correct regular expression regardless of the order in which we rip states, but certain orders will make our life easier. Because state 4 has no transitions to other states, it is easy to rip state 4.



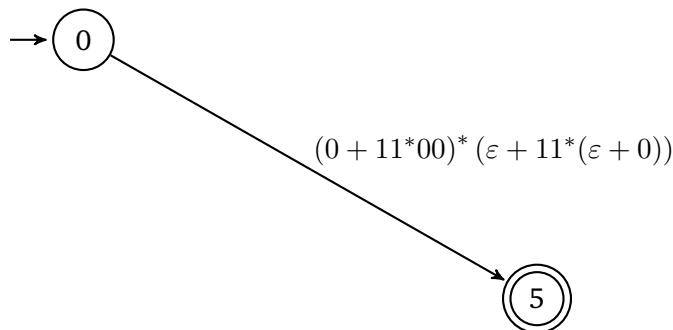
Because state 3 only has one transition in and two transitions out, we rip it next. (State 2 might also be reasonable to rip here, but we prefer state 3 because it lacks a self-loop.)



Now, state 1 has two transitions coming in and two going out. State 2 has one transition coming in and two going out, so ripping state 2 is easier.



Finally, ripping state 1 gives us the following.

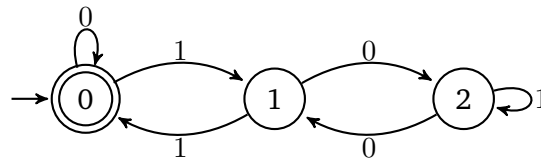


Hence,  $(0 + 11^*00)^* (\epsilon + 11^*(\epsilon + 0))$  is a regular expression for  $A$ . ■

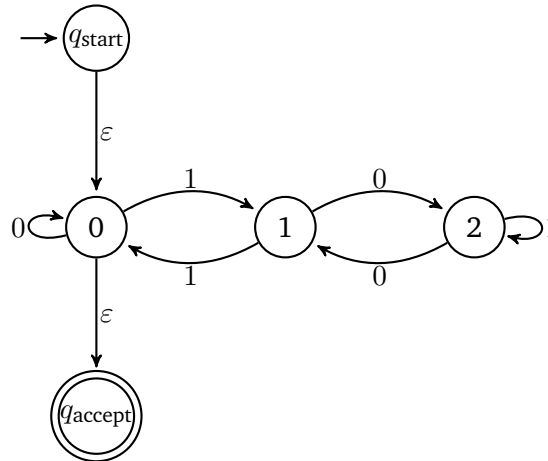
- (c)  $\{w \mid w \text{ encodes a multiple of 3 in binary}\}$

**Solution:** Let  $A$  be the language above. We take the same approach as in part (b). To design a DFA for  $A$ , we consider how reading the next bit changes the number encoded by the input. Note that if we have read an input that encodes  $n$ , reading a 0 shifts all the bits left by one place and therefore changes the encoded number from  $n$  to  $2n$ . If we read a 1, then the encoded number changes from  $n$  to  $2n + 1$ .

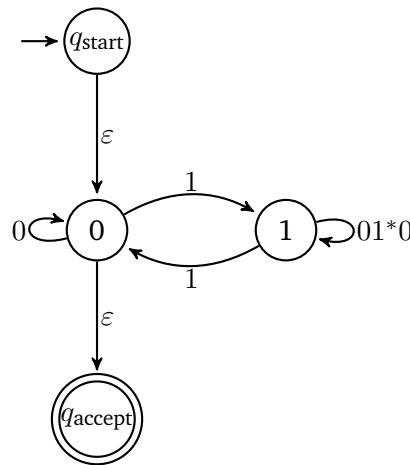
Our DFA uses three states  $\{0, 1, 2\}$ , one for each of the three possible remainders when the encoded number  $n$  is divided by 3. The machine works because knowing the current remainder is enough to determine the new value of the remainder after reading an input bit. A DFA that recognizes  $A$  appears below.



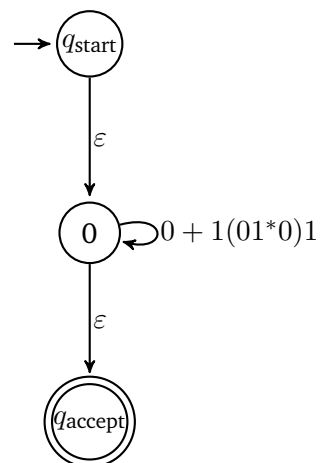
We convert to a GNFA.



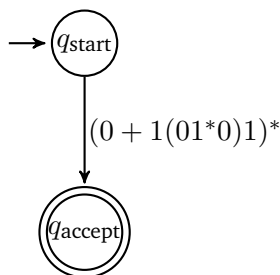
We rip state 2.



Next, it is state 1's turn.



Finally, we rip state 0.



Therefore  $(0 + 1(01^*0)1)^*$  is a regular expression for the binary encodings of multiples of three. ■

6. [30 points; 15 per part] For every  $a \in \Sigma$ , let  $\#a(w)$  denote the number of  $a$  symbols appearing in  $w$ . Use the pumping lemma to prove that the following languages over  $\Sigma = \{a, b, c\}$  are not regular:

- (a)  $\{xycy \mid x, y \in \{a, b\}^* \text{ and } \#a(x) = \#b(y)\}$

**Solution:** Let  $A$  be the language above. We show how to win the pumping lemma game for  $A$ . (In this pumping lemma game, we will use  $w = tuv$  instead of the traditional  $w = xyz$  to avoid any possible confusion with use of  $x$  and  $y$  in defining the language  $A$ .)

- i. The adversary chooses an integer  $p \geq 0$ .
- ii. We choose  $w = a^p cb^p$ . Note that  $|w| \geq p$  and  $w \in A$ .
- iii. The adversary splits  $w$  into 3 parts,  $w = tuv$ , where  $|tu| \leq p$  and  $|u| \geq 1$ . Note that because  $|tu| \leq p$ , the string  $tu$  consists entirely of  $a$ 's.
- iv. Consider the string  $tu^i v$ . Because  $u$  contains only  $a$ 's, we know that  $tu^i v = a^{p+(i-1)|u|} cb^p$ . We choose  $i = 0$ , so that  $tu^0 v = tv = a^{p-|u|} cb^p$ . Because  $u$  is not the empty string, we have that  $tu^0 v \notin A$ , and we win.

Because we have a winning strategy for the pumping lemma game,  $A$  is not regular. ■

- (b)  $\{w \mid w = w^R\}$ , i.e., the set of all *palindromes*

**Solution:** Let  $A$  be the language above. We show how to win the pumping lemma game for  $A$ .

- i. The adversary chooses an integer  $p \geq 0$ .
- ii. We choose  $w = a^p ba^p$ . Note that  $|w| \geq p$  and  $w \in A$ .
- iii. The adversary splits  $w$  into 3 parts,  $w = xyz$  where  $|xy| \leq p$  and  $|y| \geq 1$ . Note that because  $|xy| \leq p$ , the string  $xy$  consists entirely of  $a$ 's.
- iv. Consider the string  $xy^i z$ . Because  $y$  contains only  $a$ 's, we know that  $xy^i z = a^{p+(i-1)|y|} ba^p$ . We choose  $i = 0$ , so that  $xy^0 z = xz = a^{p-|y|} ba^p$ . Because  $y$  is not the empty string, we see that  $xz$  has fewer  $a$ 's to the left of the single  $b$  than to the right. Therefore  $xz$  is not a palindrome, so  $xz \notin A$  and we win.

Because we have a winning strategy for the pumping lemma game,  $A$  is not regular. ■

7. [Honors; 20 points] Prove that the language  $\{a^p \mid p \text{ is prime}\}$  is not regular.

**Solution:** Let  $A$  be the language above. We show how to win the pumping lemma game for  $A$ .

- The adversary chooses an integer  $p \geq 0$ .
- Let  $n$  be a prime number that is at least as large as  $p$ . We choose  $w = a^n$ . Note that  $|w| \geq p$  and  $w \in A$ .
- The adversary splits  $w$  into 3 parts,  $w = xyz$  where  $|xy| \leq p$  and  $|y| \geq 1$ .
- Consider the string  $xy^iz$ . We know that  $xy^iz = a^{n+(i-1)|y|}$ . We choose  $i = n + 1$ , so that

$$xy^{n+1}z = a^{n+((n+1)-1)|y|} = a^{n+n|y|} = a^{n(1+|y|)}.$$

Therefore the length of  $xy^iz$  is  $n(1 + |y|)$ , which we claim is not a prime number. Indeed, because  $|y| \geq 1$ , we see that  $1 + |y| \geq 2$ . Because  $n$  is a prime number, we also know that  $n \geq 2$ . Therefore  $n(1 + |y|)$  is the product of two integers, both of which are at least 2, and so  $n(1 + |y|)$  is not prime as claimed.

It follows that  $xy^{n+1}z \notin A$  and therefore we win.

Because we have a winning strategy for the pumping lemma game,  $A$  is not regular. ■